Exhibit 4

Tornado Cash Gas Analysis

Description

This gas analysis was performed in order to determine the usage of the Tornado Cash User Interface (UI) and Command Line Interface (CLI). The gas limit and the gas ratio ($\frac{g_{used}}{g_{limit}}$) were retrieved for deposits into Tornado Cash on the Ethereum blockchain for Ether for the time period between September 1, 2020 – August 8, 2022. The gas limit and ratio were then compared against the gas limit created by the UI code and the gas limit and ratio created by the CLI code for a number of different time periods to determine the estimated percentage of UI and CLI usage. It should be noted that UI and CLI determinations are estimated and not exact. It is possible that a user not using the CLI or UI sets a gas limit value that happens to fall into the range of determined UI and CLI values. Therefore, this should be considered an approximation to try to quantify the UI and CLI usage from figures 1 and 2.

Range Calculation Methodology

The UI and CLI both had distinct methods for calculating gas utilization for deposits. The first time period for both the UI and CLI (covered more in Calculated Ranges and Time Periods) was normally hardcoded to a specific value. After this time period, the gas limit was calculated to be a multiple of the estimated gas (1.3) for the CLI or had a specific amount (50,000) added to it for the UI. The starting point for finding the ranges was plotting the gas limit against time as shown in figure 1 and the gas ratio against time shown in figure 2.

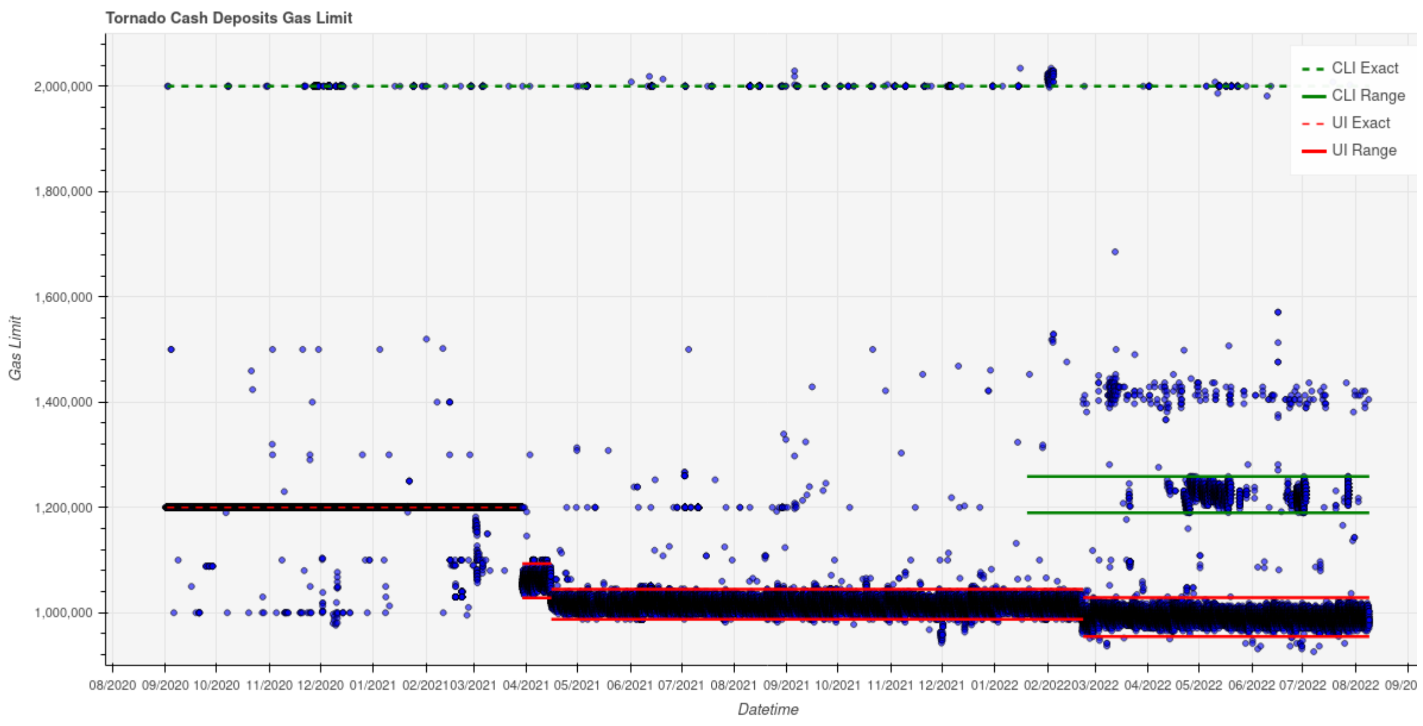


Figure 1. Gas limit over time for Tornado Cash deposits. Note the distinct bands for the UI shown in red lines towards the bottom.

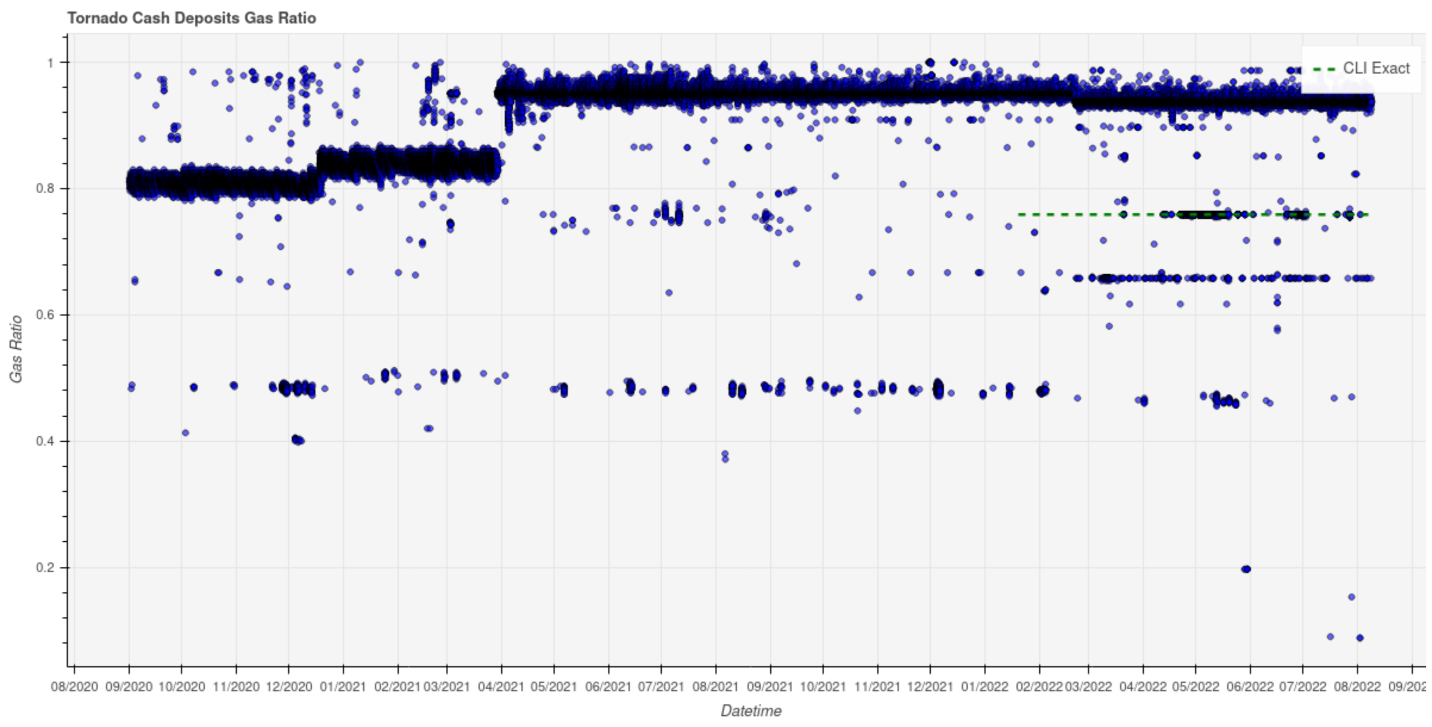
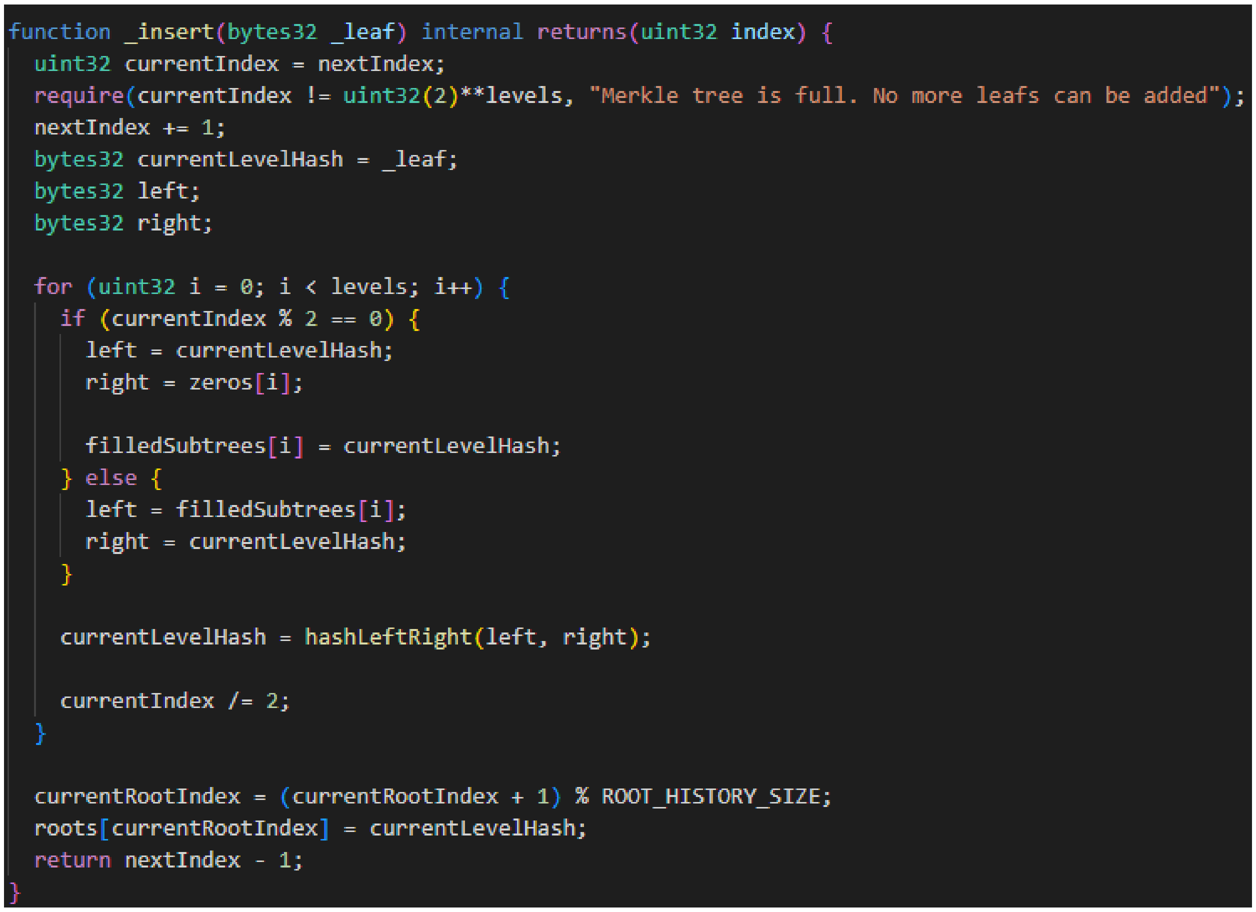


Figure 2. Gas ratio over time for Tornado Cash deposits. The dotted green line is at exactly 0.759.

The reason why a band is created during gas estimation is due to the way the merkle tree is constructed when a deposit occurs (specifically the _insert function). When a deposit is inserted into the merkle tree, it must reconstruct the entire tree, using cached values for any values to the left of the current index and the zero value (precomputed ahead of time) for any value to the right of the current node. If the current index is divisible by 2, the current hash value must be stored in the cache using the SSTORE opcode which has a gas cost of 5,000. This SSTORE is not incurred if the current index is odd, resulting in lower gas costs. This loop is done for all 20 levels of the Tornado Cash Merkle Tree, which means that there can be a large range of potential gas values. However, most deposits during a time period are confined between a fairly small number of indexes meaning the discrepancies aren't quite as wide.



```
function _insert(bytes32 _leaf) internal returns(uint32 index) {
  uint32 currentIndex = nextIndex;
  require(currentIndex != uint32(2)**levels, "Merkle tree is full. No more leafs can be added");
  nextIndex += 1;
  bytes32 currentLevelHash = _leaf;
  bytes32 left;
  bytes32 right;

  for (uint32 i = 0; i < levels; i++) {
    if (currentIndex % 2 == 0) {
      left = currentLevelHash;
      right = zeros[i];

      filledSubtrees[i] = currentLevelHash;
    } else {
      left = filledSubtrees[i];
      right = currentLevelHash;
    }

    currentLevelHash = hashLeftRight(left, right);

    currentIndex /= 2;
  }

  currentRootIndex = (currentRootIndex + 1) % ROOT_HISTORY_SIZE;
  roots[currentRootIndex] = currentLevelHash;
  return nextIndex - 1;
}
```

Figure 3. Code for inserting into Tornado Cash Merkle tree from: https://vscode.blockscan.com/ethereum/0x12D66f87A04A9E220743712cE6d9bB1B5616B8Fc

User Interface

There appear to be three distinct regions after the gas limit calculation is changed from an exact value to an estimation on or about 3/29/2021. These regions are likely due to gas estimation changes from either code changes, smart contract architecture changes (changing the amount of gas used by a transaction), or EVM changes (hard forks). To find these ranges, thousands of deposits were simulated on a fork of Ethereum mainnet from the relevant block number for the specific time period using Foundry's anvil software and an Infura RPC endpoint.[1] This was executed using custom code matching the relevant deposit code from the period. The code was made to closely match the relevant gas estimation code from the UI which remained more or less the same during the relevant period.

The program was made to match the UI as closely as possible, including using the same version of the web3 library and using the same RPC provider (Infura). On the smart contract side, the correct router contract was chosen for the specific time period. Additionally, each of the pools was simulated, from the approximate starting merkle tree index in the time period to either the approximate ending merkle tree index for that time period or approximately 1,000 deposits,

[1] https://book.getfoundry.sh/anvil/

whichever came first. Finally, the encryptedNote functionality was simulated for the final time period only to make the results more conservative. The minimum and maximum gas limit was then taken as the UI band range for each respective period.

Command Line Interface

After the initial hardcoded value of 2,000,000 for the deposit gas, the gas limit was calculated as estimated gas used * 1.3. The theoretical gas ratio can then be calculated as:

$$g_{limit} = 1.3 * g_{used}$$

$$g_{ratio} = \frac{g_{used}}{g_{limit}} = \frac{g_{used}}{1.3 * g_{used}} = \frac{1}{1.3}$$

Equation 1. Calculating theoretical gas ratio where g_{used} is the gas used by the transaction (or estimated to be used) and g_{limit} is the gas limit for the transaction.

Therefore, the theoretical gas ratio should be equal to 0.769, but in practice the gas ratio is closer to 0.759 as can be seen from figure 2. This could be due to the slight inaccuracies in the gas estimation algorithm or other factors. Therefore, any deposit with a gas ratio between 0.758 and 0.760 was categorized as the CLI, to account for a very small range of deviance from 0.759.

Calculated Ranges and Time Periods

User Interface

12/16/2019-3/29/2021

Based on the code from git commit c2924dade5222f01f4f622540a9da705da7d55d9 on 12/16/2019, the gas limit for a deposit to the UI was exactly 1,100,000 + 100,000 = 1,200,000 gas as shown in figure 3. Therefore, anything in this time range that was exactly 1,200,000 gas was considered to be the UI. No transactions were simulated for the time period since it was an exact amount.

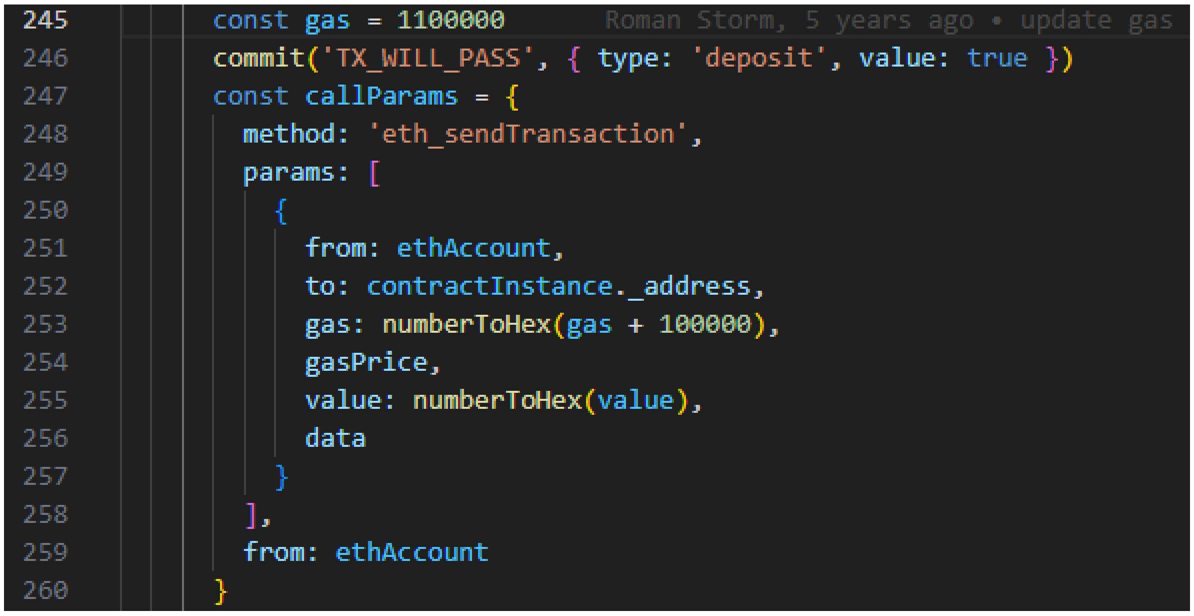

```
const gas = 1100000        Roman Storm, 5 years ago • update gas
commit('TX_WILL_PASS', { type: 'deposit', value: true })
const callParams = {
  method: 'eth_sendTransaction',
  params: [
    {
      from: ethAccount,
      to: contractInstance._address,
      gas: numberToHex(gas + 100000),
      gasPrice,
      value: numberToHex(value),
      data
    }
  ],
  from: ethAccount
}
```

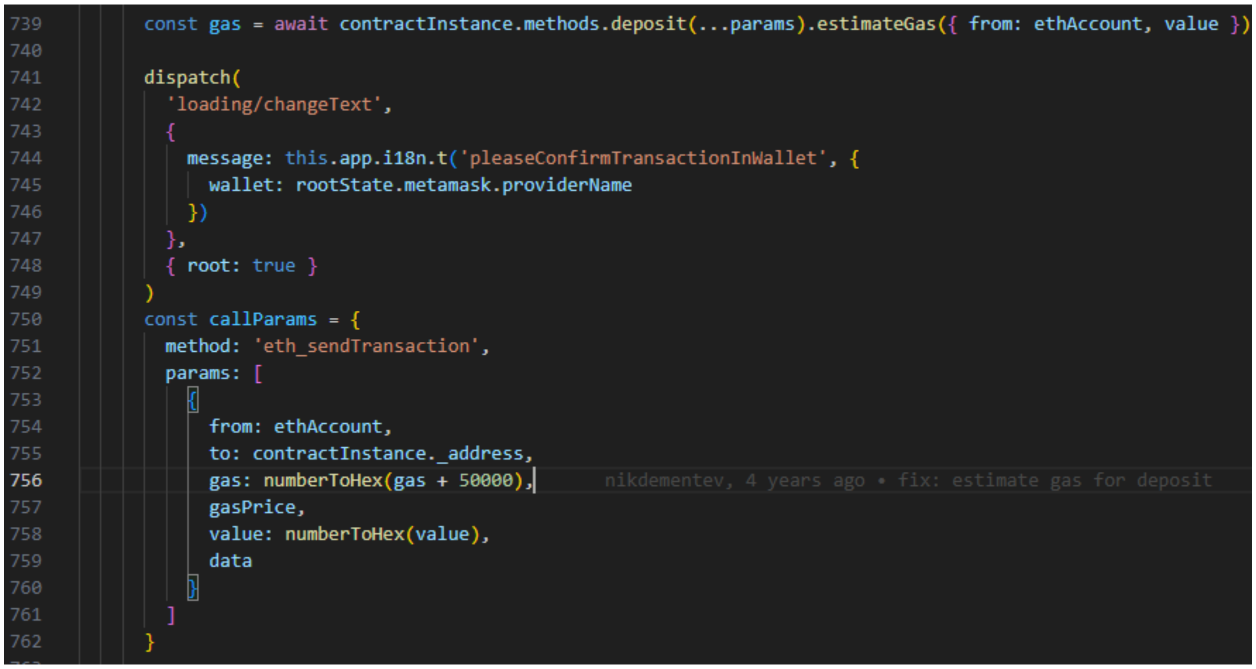
Figure 4. Gas calculation for code found in store/mixer.js.

3/29/2021-4/15/2021

Simulation details:

- Web3 Version: 1.3.3
- Proxy (Router) contract: 0x722122df12d4e14e13ac3b6895a86e8414 5b6967
- Range: 1,027,998-1,092,986

On 3/29/2021 at approximately 13:31, the git commit 95243a9c8d05ac6b04f8595abe153396f2ed39e3 was deployed to the website. This git commit changed the gas limit estimation from an exact 1.2 million to an estimation of gas plus 50,000 as seen in figure 4. The resulting band was from 1,027,998-1,092,986.



```
      const gas = await contractInstance.methods.deposit(...params).estimateGas({ from: ethAccount, value })

      dispatch(
        'loading/changeText',
        {
          message: this.app.i18n.t('pleaseConfirmTransactionInWallet', {
            wallet: rootState.metamask.providerName
          })
        },
        { root: true }
      )
      const callParams = {
        method: 'eth_sendTransaction',
        params: [
          {
            from: ethAccount,
            to: contractInstance._address,
            gas: numberToHex(gas + 50000),        nikdementev, 4 years ago • fix: estimate gas for deposit
            gasPrice,
            value: numberToHex(value),
            data
          }
        ]
      }
```

Figure 5. Gas calculation for sendDeposit inside mixer.js.

4/15/2021 – 2/21/2022

Simulation details:

- Web3 Version: 1.5.2 (Changed from 1.3.3 on or about 8/31/2021)
- Proxy (Router) contract: 0x722122df12d4e14e13ac3b6895a86e84145b6967
- Range: 987,298-1,044,586

On or around 4/15/2021, the Berlin hard fork to the Ethereum blockchain was introduced.[2] This hard fork introduced EIP-2929, which increased gas costs for the first address lookup for DELEGATECALL and EXTCODESIZE op codes from 700 to 2,600 gas.[3] However, the data is then stored in a cache for subsequent calls so subsequent calls only cost 100 gas. The deposit function inside of the Tornado Cash pools utilizes the function hashLeftRight to calculate the Merkle tree. This hashLeftRight function performs two EXTCODESIZE calls to the hasher function at 0x83584f83f26af4edda9cbe8c730bc87c364b28fe and two DELEGATECALL calls to the same address. This results in a similar amount of gas used for the first time this function is called, followed by a savings of 2,400 gas every time the function is called. The hashLeftRight function is called 20 times a transaction (because there are 20 levels in the Merkle tree), resulting in a gas savings of around 45,600 gas per transaction. This savings amount is reduced from other parts of the transaction costing more gas (like storage loads), but the net outcome is less total gas used in the transaction. This causes the estimated gas limit for deposit transactions to be reduced as well. The gas limit was estimated from 987,298-1,044,586 for this time period.

[2] https://medium.com/ethereum-cat-herders/the-berlin-upgrade-overview-2f7ad710eb80
[3] https://eips.ethereum.org/EIPS/eip-2929

2/21/2022 – 8/8/2022

Simulation details:

- Web3 Version: 1.5.2
- Proxy (Router) contract: 0xd90e2f925da726b50c4ed8d0fb90ad053324f31b
- Range: 954,878-1,028,948

The last band occurs because the smart contract architecture was changed around this time to incorporate a new router, relayer registry, and remove anonymity mining amongst other features. This caused the range to vary slightly. For this range, the gas limit was estimated to be from 954,878-1,028,948.

Datetime Start	Datetime End	Range Start	Range End
2020-09-01 00:00:00	2021-03-29 13:31:34	1,200,000	1,200,000
2021-03-29 13:31:35	2021-04-15 14:59:59	1,027,998	1,092,986
2021-04-15 15:00:00	2022-02-21 23:59:59	987,298	1,044,586
2022-02-22 00:00:00	2022-08-08 23:59:59	954,878	1,028,948

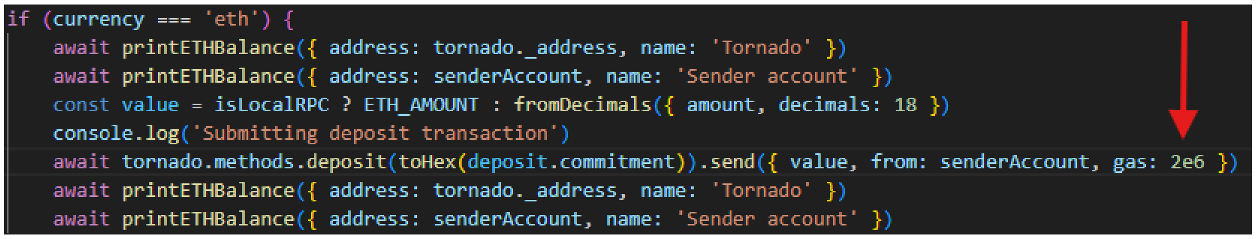
Table 1. Date ranges and gas limits for UI calculation. Note that all datetimes are in UTC and the range values are gas limit values.

Command Line Interface

The command line interface estimation varies from the UI estimation because the CLI could be downloaded and used at any time, whereas users of the frontend website normally just used the latest deployed version.

5/21/2020 – 8/8/2022

The first commit (2e4d59aed06e2c58f79e2cbe521b80b87cdeb067) of the repository established the gas limit to be 2 million. Therefore, any transaction with a gas limit exactly equal to 2 million was considered to be a CLI transaction.

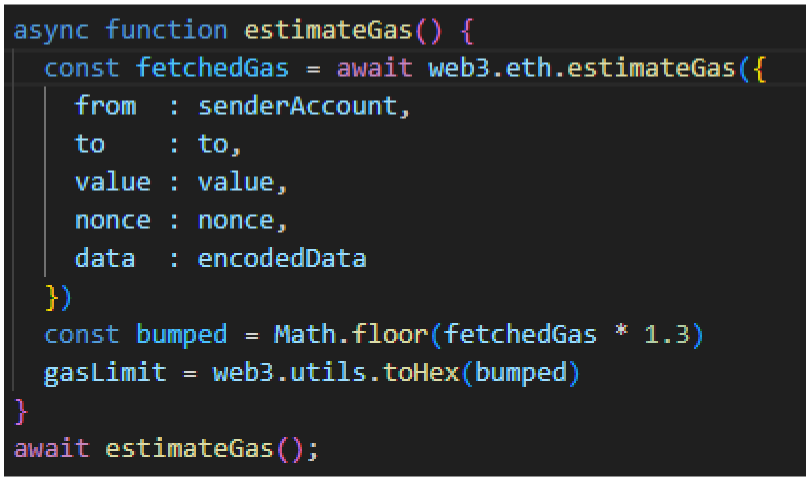


```
if (currency === 'eth') {
    await printETHBalance({ address: tornado._address, name: 'Tornado' })
    await printETHBalance({ address: senderAccount, name: 'Sender account' })
    const value = isLocalRPC ? ETH_AMOUNT : fromDecimals({ amount, decimals: 18 })
    console.log('Submitting deposit transaction')
    await tornado.methods.deposit(toHex(deposit.commitment)).send({ value, from: senderAccount, gas: 2e6 })
    await printETHBalance({ address: tornado._address, name: 'Tornado' })
    await printETHBalance({ address: senderAccount, name: 'Sender account' })
```

Figure 6. Gas calculation from initial commit

1/20/2022 – 8/8/2022

The next change to the gas limit came in the pull request: 4b13d53cab4cd6bcfa038719862c0e55a589baf4 which was merged into the main repository on 1/20/2022 by Storm. This set the theoretical gas ratio to be 0.759 as discussed in Range Calculation Methodology.



```
async function estimateGas() {
  const fetchedGas = await web3.eth.estimateGas({
    from  : senderAccount,
    to    : to,
    value : value,
    nonce : nonce,
    data  : encodedData
  })
  const bumped = Math.floor(fetchedGas * 1.3)
  gasLimit = web3.utils.toHex(bumped)
}
await estimateGas();
```

Figure 7. Gas limit estimation from commit

Datetime Start	Datetime End	Range Start	Range End
2020-09-01 00:00:00	2022-08-08 23:59:59	2,000,000	2,000,000
2022-01-20 00:00:00	2022-08-08 23:59:59	0.758	0.760

Table 2 . Date ranges and gas limits for CLI calculation. Note that all datetimes are in UTC and the first range values are gas limits and the second range values are gas ratios.

Results

The usage of the UI, CLI, and unknown usage can be found in table 3.

Category	Count	Percentage
UI	135,120	96.16
CLI	3,908	2.78
Unknown	1,486	1.06

Table 3. Results from UI and CLI calculations

Comparison of Simulated Values Against Actual Gas Values

The actual gas limits for a given index were compared against the simulated gas limits to see whether there was any crossover with actual data. The actual gas values were pulled from Dune Analytics and then matched against simulated transactions on the Tornado Cash pool and leaf index number. If the actual transaction had used an encrypted note and the simulated transaction had not, the transaction was excluded from the analysis and vice-versa. Adding an encrypted note increases the gas by at least 2,000 gas which would skew the results. Of the 14,597 simulated transactions, 9,412 met the encryptedNote criteria.

The results showed very good overlap with actual transactions, with over 62% matching the gas value on chain exactly. An additional ~21% were within 100 gas of the simulated transaction which can be considered a near exact match. This means that 83% of on chain transactions had a near or exact match with the simulated transactions. The results of the verification can be seen in table 4.

Category	Count	Percentage
Exact	5,857	62.23
Less_than_100_difference	1,952	20.74
Over_100_difference	1,603	17.03

Table 4. On-chain vs simulated gas limits.

The simulation gas limits showed close alignment with gas limits submitted on-chain. The actual on-chain values can vary from the simulated for several reasons including differences between how gas values were estimated by Infura nodes at the time, small changes in the architecture or code which could have changed the values being calculated which was not visible on the graph, and users not using the UI. The difference in gas-estimation can be seen vividly for the period between 3/29/2021 and 4/15/2021 as this time period was pre-Berlin hard fork. The simulations were done post-Berlin hard fork (although it is unknown if Infura used a pre-Berlin hard fork environment for the estimatedGas caclculation) and a total of 574 simulations differed by over 100 gas. This makes up over a third of the total differences over 100 gas, despite being less than a quarter of the total transactions analyzed. This comparison helps further reinforce the validity of the gas range bounds created from the simulations.

Conclusion

Although care was taken to try to control as many variables as possible, this should be considered an approximation of the number of users who used the UI or CLI. This work only focused on Ether pools on the Ethereum blockchain and the usage of the UI or CLI on other chains or assets could vary, although Ether on Ethereum was one of the most popular during the relevant period. Additionally, it is possible that a developer set a gas limit inside one of these ranges, and there is no way to currently separate these developers from other users using this approach.